

Univerzitet u Sarajevu

Elektrotehnički fakultet u Sarajevu

Računarstvo i informatika

# eParking

## Dokumentacija modela

### Predmet: Objektno orijentisana analiza i dizajn

Članovi grupe: Milica Đokic, Faris Demir, Amar Bešlagić

## **Opis teme**

eParking je aplikacija koja omogućava korisniku pronalaženje najbližeg slobodnog parking mesta korištenjem Google mape, kao i prikazivanje rute do tog parking mesta. Osim toga omogućena je i online registracija korisnika, koju odobrava vlasnik. Registracijom korisnici stiču članstvo i rezervisano parking mjesto. Prilikom ulaska korisnika na parking pokreće se timer koji se zaustavlja kada korisnik napušta parking, i na osnovu toga se formira račun. Omogućeno je i unošenje novih parkinga u sistem.

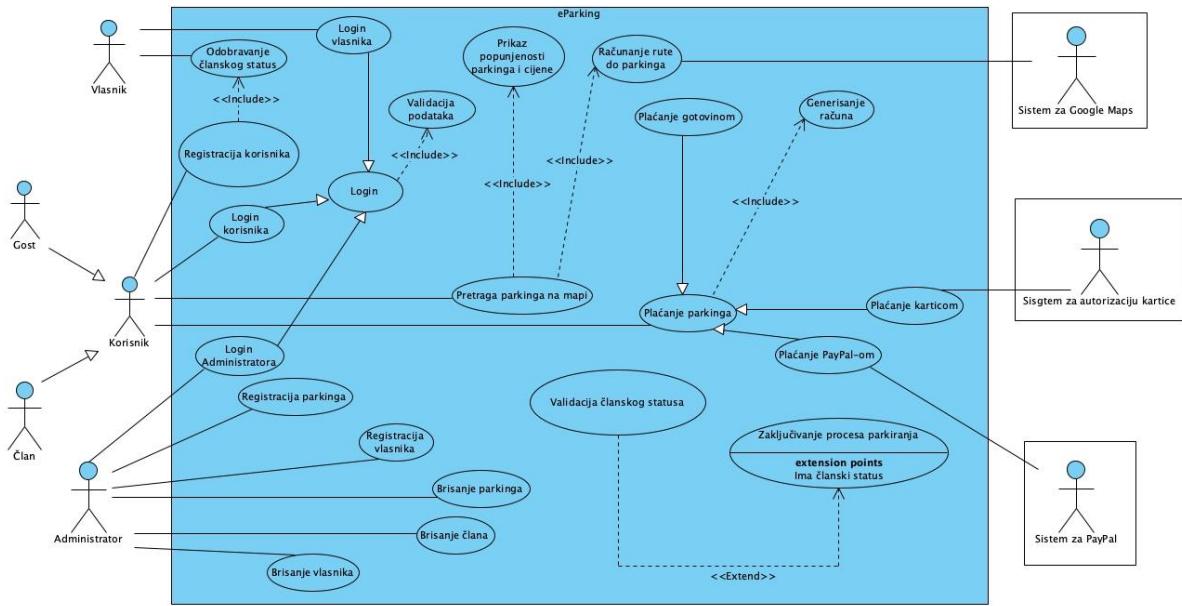
## **Funkcionalnosti**

- Mogućnost online registracije
- Mogućnost pronalaženja najbližeg slobodnog parking mesta preko Google mape, kao i računanje rute do tog parking mesta
- Mogućnost dobijanja informacija o svim parkinzima na mapi (lokacija, broj slobodnih mjesta)
- Mogućnost registracije novih parkinga
- Pregled podataka i akcija unutar korisničkog računa
- Mogućnost plaćanja gotovinom, karticom ili PayPal-om.

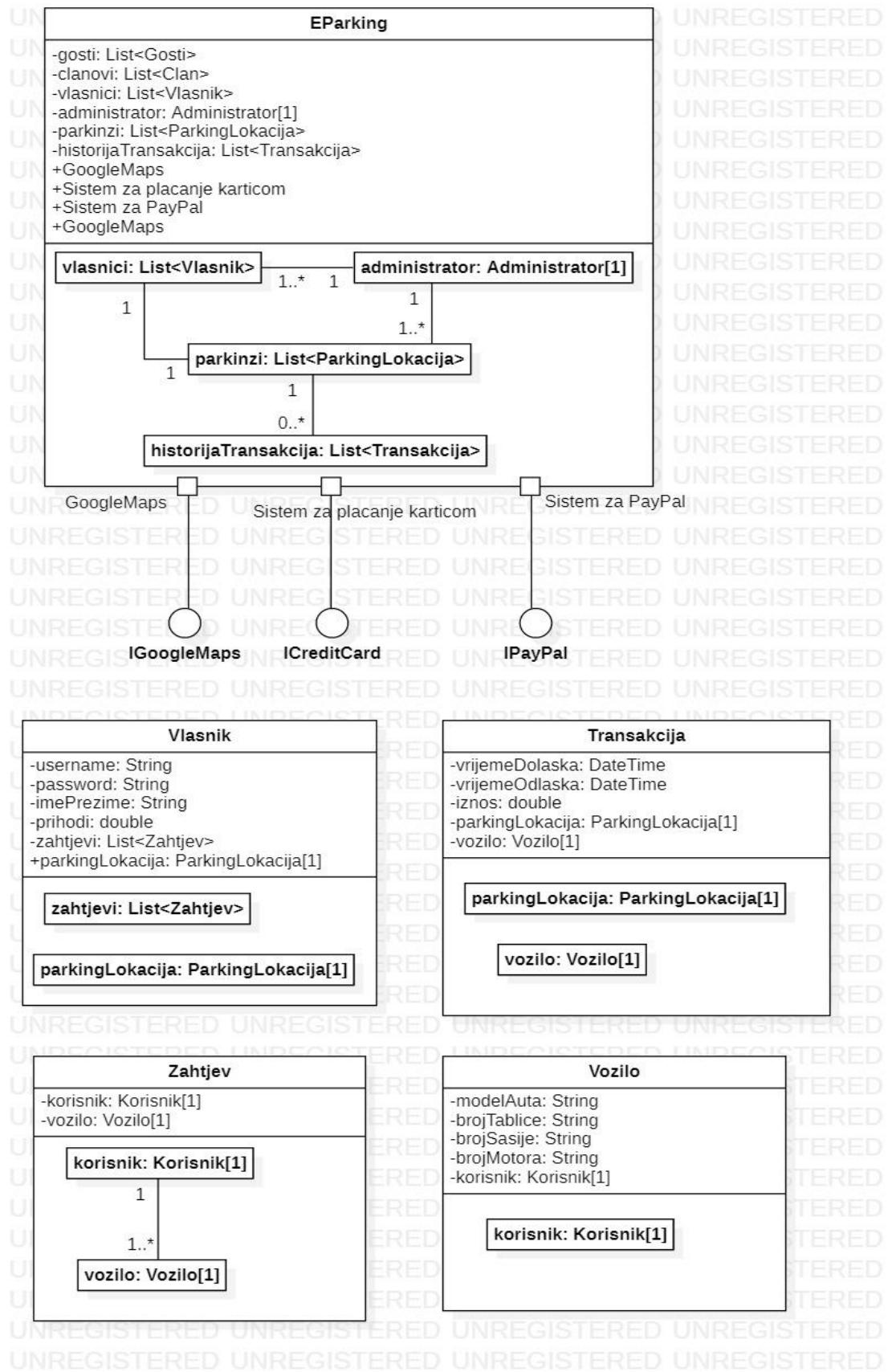
## **Akteri**

1. Korisnik - vlasnik prevoznog sredstva koji želi da koristi eParking. Podjela korisnika:
  - registrovani korisnici(članovi) – korisnici koji imaju plaćenu mjesečnu ili godišnju članarinu i imaju rezervisano parking mjesto na izabranom parkingu
  - gosti – korisnici koji nisu registrovani i koji vrše upлатu svaki put kada dođu na parking
2. Vlasnik – nadgleda rad eParkinga, odobrava ili odbija zahtjeve za registraciju novih korisnika
3. Administrator – registruje nove parkinge i briše neke već postojeće, mijenja podatke o parkinzima, i po potrebi ažurira i ostale podatke
4. Sistem za autorizaciju kartice – kontroliše korisničke uplate, te vrši njihovu validaciju
5. Sistem za Google Maps – prikazuje mapu sa registrovanim parkinzima, te računa rutu do izabranog parkinga i prikazuje vrijeme potrebno vozilu da stigne do odredišta
6. Sistem za PayPal – kontroliše korisničke uplate, te vrši njihovu validaciju

## Dijagram slučajeve upotrebe (Use case diagram)



## Dijagram slozene strukture



## Scenariji

### Scenario za plaćanje parkinga.

Naziv	Naplata parkinga
Opis	Korisnik plaća parking ili članarinu
Preduvjeti	Login korisnika, odobrena autorizacija kartice ili PayPal računa
Posljedice – uspješan završetak	Plaćen parking
Posljedice – neuspješan završetak	Parking nije plaćen
Primarni akteri	Korisnik
Ostali akteri	Sistem za autorizaciju kartice i PayPal računa
Glavni tok	Korisnik bira da li želi da plati parking na osnovu vremena provedenog na istom ili da uplati mjesecnu ili godišnju članarinu

Za uspješan završetak:

Korisnik	Sistem za autorizaciju kartice ili PayPal računa	Sistem eParking
1. Otvaranje prozora za plaćanje		
2. Odabir da li se plaća parking za vrijeme provedeno na njemu ili se plaća mjesecna ili godišnja članarina		
3. Odabir sredstva za plaćanje		
	4. Obavijest o uspješnoj autorizaciji kartice ili PayPal računa	
		5. Generisanje računa

Za neuspješan završetak:

Korisnik	Sistem za autorizaciju kartice ili PayPal računa
1. Otvaranje prozora za plaćanje	
2. Odabir da li se plaća parking za vrijeme provedeno na njemu ili se plaća mjesecna ili godišnja članarina	
3. Odabir sredstva za plaćanje	
	4. Obavijest o neuspješnoj autorizaciji kartice ili PayPal računa
5. Povratak na početni prozor	

## Scenario za registraciju parkinga

Naziv	Registracija parkinga
Opis	Administrator registruje novi parking
Preduvjeti	/
Posljedice – uspješan završetak	Uspješno registrovan novi parking
Posljedice – neuspješan završetak	/
Primarni akteri	Administrator
Ostali akteri	/
Glavni tok	Administrator registruje novi parking i dodaje ga u bazu podataka sa pratećim sadržajem (kapacitet parkinga, cijena mjesecne članarine, cijena godišnje članarine, cijena po satu parkinga)

Za uspješan završetak:

Administrator
1. Unos svih informacija o parkingu u bazu podataka
2. Objavljivanje novog parkinga na mapi

## Scenario za parkiranje

Naziv	Parkiranje korisnika
Opis	Korisnik pronalazi najbliže parking mjesto na mapi i parkira vozilo
Preduvjeti	Uspješan login
Posljedice – uspješan završetak	Korisnik pronalazi parking mjesto
Posljedice – neuspješan završetak	Korisnik ne pronalazi parking mjesto
Primarni akteri	Korisnik
Ostali akteri	Sistem za Google Maps
Glavni tok	Korisnik na Google mapi pronalazi da li ima slobodnih parking mjesta. Ukoliko ima, korisnik odlazi na izabранo parking mjesto. Prilikom ulaska na parking pokreće se timer. Ukoliko korisnik nema već plaćenu članarinu, na osnovu vremena provedenog na parkingu se formira račun za naplatu.

Za uspješan završetak:

Gost	Sistem eParking	Sistem za Google Maps
1. Pristup Google mapi		
2. Traženje najbližeg slobodnog parking mesta		

	3. Slanje potvrde da je slobodno parking mjesto pronađeno	
		4. Računanje rute do parkinga
	5. Registrovanje ulaska na parking i pokretanje timera	
	6. Registrovanje izlaska sa parkinga i zaustavljanje timera	
	7. Generisanje računa	

Za neuspješan završetak:

Gost	Sistem eParking
1. Pristup Google mapi	
2. Traženje najbližeg slobodnog parking mjesta	
	3. Slanje potvrde da slobodno parking mjesto nije pronađeno

Za uspješan završetak:

Član	Sistem eParking	Sistem za Google Maps
1. Pristup Google mapi		
		2. Računanje rute do izabranog parking mjesta
	3. Registrovanje ulaska na parking	
	4. Validacija članskog statusa	
	5. Registrovanje izlaska sa parkinga	
	6. Izdavanje računa ukoliko je korisnik bio na nerezervisanom mjestu	

## Scenario za registraciju korisnika

Naziv	Registracija korisnika
Opis	Korisnik navodi lične podatke i odabire parking mjesto
Preduvjeti	/
Posljedice - uspješan završetak	Kreiran korisnički nalog
Posljedice - neuspješan završetak	Korisnički nalog nije kreiran
Primarni akteri	Gost
Ostali akteri	Vlasnik
Glavni tok	Korisnik ispunjava formu za registraciju koja zahtjeva osnovne podatke o njemu samom te podatke o vozilu. Ima pravo izbora jednog parking mesta na parkingu po njegovoj želji. Nakon predavanja registracijske forme, vlasnik je može odobriti ili odbiti te slanjem email-a obaviještava korisnika o odluci.

Za uspješan završetak:

GOST	VLASNIK	ADMINISTRATOR
1. Popunjavanje ličnih podataka		
2. Popunjavanje podataka o vozilu		
3. Izbor parking mesta na željenom parkingu		
	4. Odobravanje prijave za članarinu	
		5. Kreiranje korisničkog profila i unos u bazu podataka
	6. Slanje email-a potvrde	

Za neuspješan završetak:

GOST	VLASNIK
1. Popunjavanje ličnih podataka	
2. Popunjavanje podataka o vozilu	
3. Izbor parking mjesta na željenom parkingu	
	4. Odbijanje prijave za članarinu
	5. Slanje email obavijesti o nemogućnosti odobrenja članarine

## Scenario za login

Naziv	Login(prijava) korisnika
Opis	Korisnik unosi pristupne podatke i prikazuje se odgovarajući prozor
Preduvjeti	Registracija korisnika
Posljedice - uspješan završetak	Pregled account-a i omogućenih akcija
Posljedice - neuspješan završetak	Traži se ponovni unos pristupnih podataka ili registracija
Primarni akteri	Gost, Član, Vlasnik i Administrator
Ostali akteri	Sistem eParking
Glavni tok	Korisnik unosi pristupne podatke nakon čega se vrši validacija podataka, te ako su ispravni otvara se novi prozor sa informacijama o korisničkom account-u te mogućim akcijama koje su mu na raspolaganju unutar aplikacije. Ovaj prikaz nudi različite opcije ovisno o vrsti prijavljenog korisnika. Ako podaci nisu ispravni, traži se ponovni unos pristupnih podataka.

Za uspješan završetak:

GOST	SISTEM EPARKING
1. Odabir opcije Guest	
	2. Prikaz mape sa parkinzima i dodatnim opcijama

ČLAN	SISTEM EPARKING
1. Unos pristupnih podataka	
	2. Validacija podataka
	3. Prikaz account-a sa svim detaljima te mogućnost otvaranja mape

VLASNIK	SISTEM EPARKING
1. Unos pristupnih podataka	
	2. Validacija podataka
	3. Prikaz neobrađenih prijava za registraciju, stanja parkinga, ostvarenih prihoda

ADMINISTRATOR	SISTEM EPARKING
1. Unos pristupnih podataka	
	2. Validacija podataka
	3. Prikaz različitih statistika o korisnicima i sistemu te opcija ažuriranja svih aspekata aplikacije

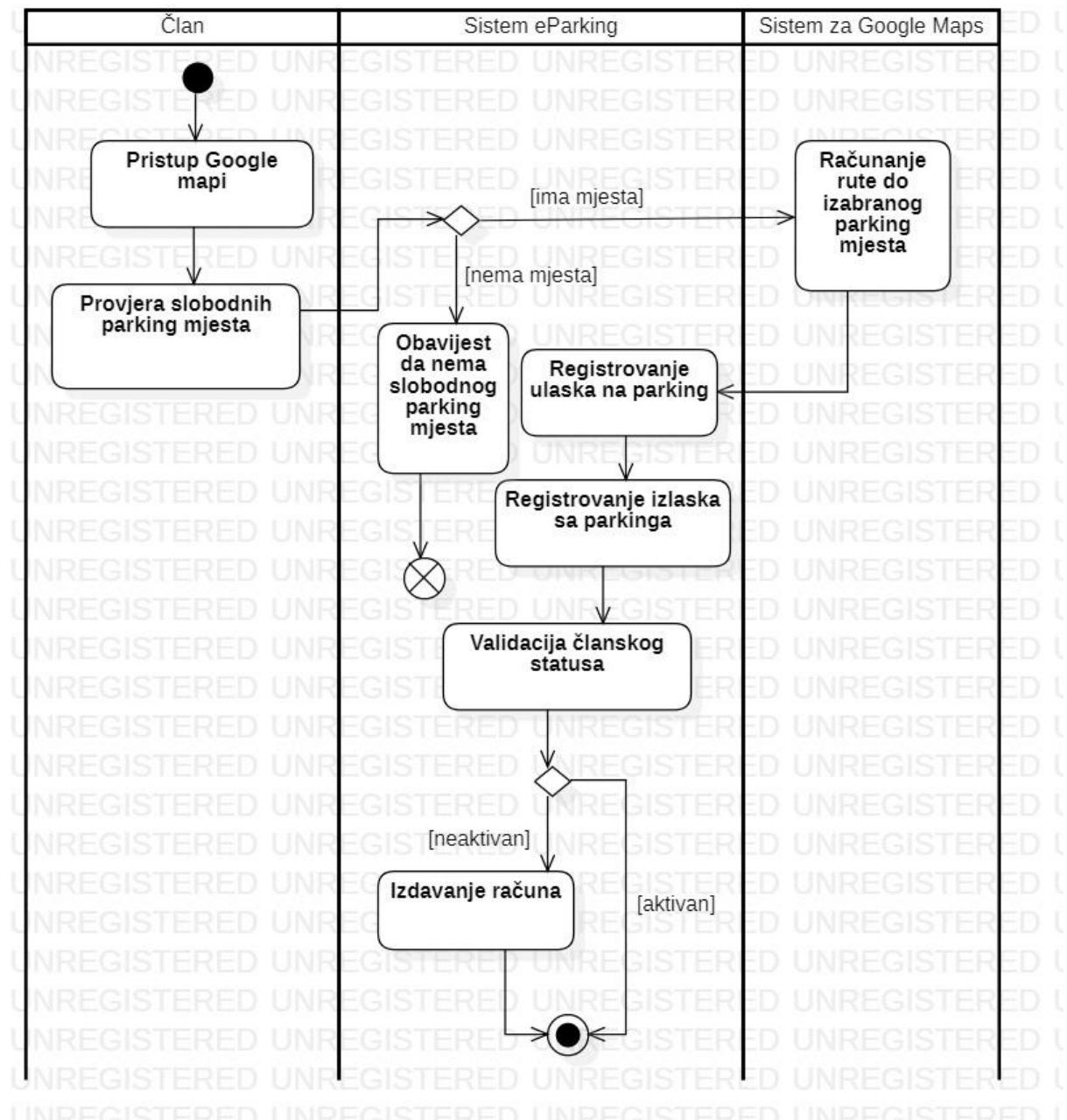
Za neuspješan završetak:

ČLAN, VLASNIK, ADMINISTRATOR	SISTEM EPARKING
1. Unos pristupnih podataka	
	2. Validacija podataka
	3. Poruka o neispravnim podacima te mogućnost ponovnog unosa

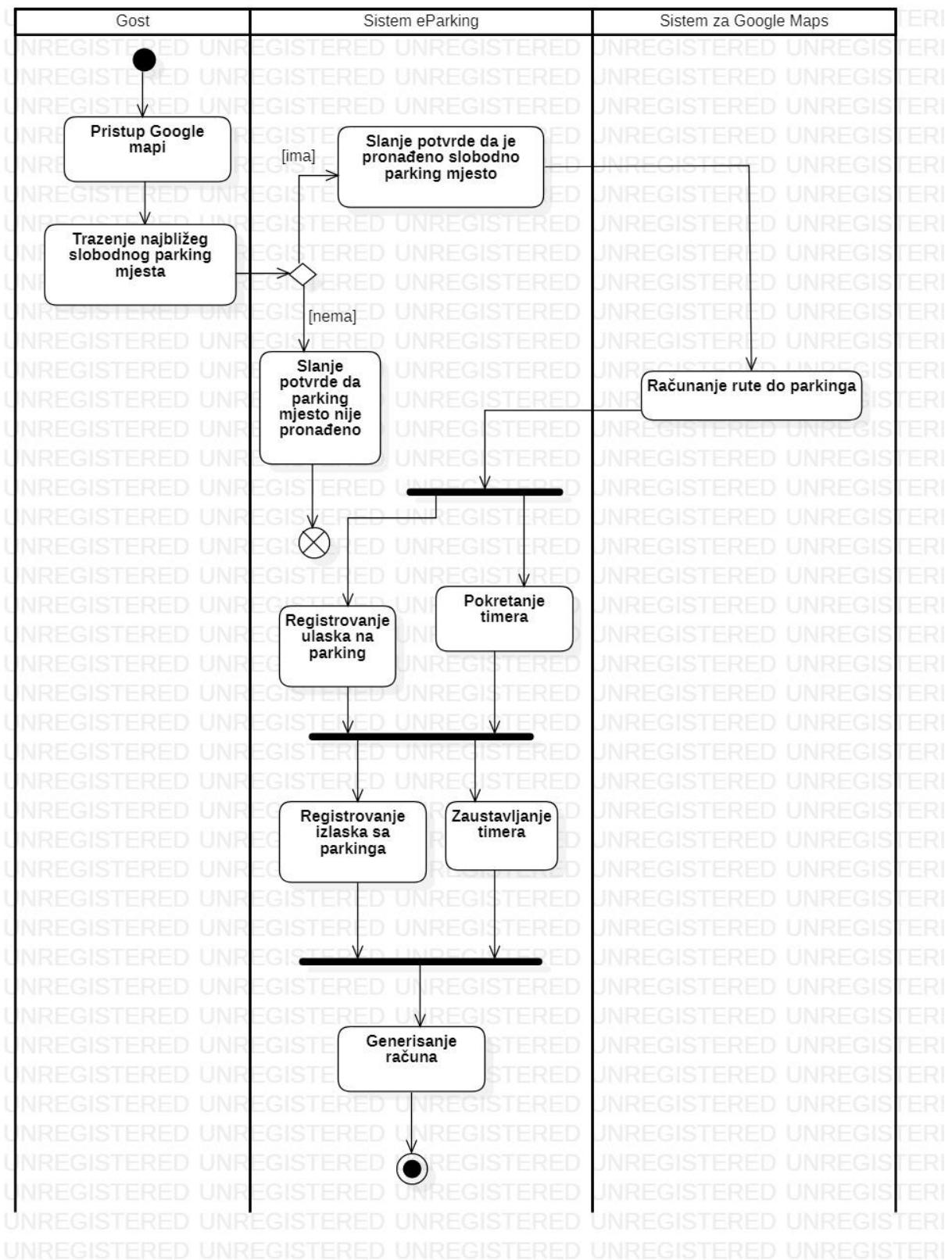
## Dijagrami aktivnosti (Activity diagrams)

U nastavku su prikazani dijagrami aktivnosti.

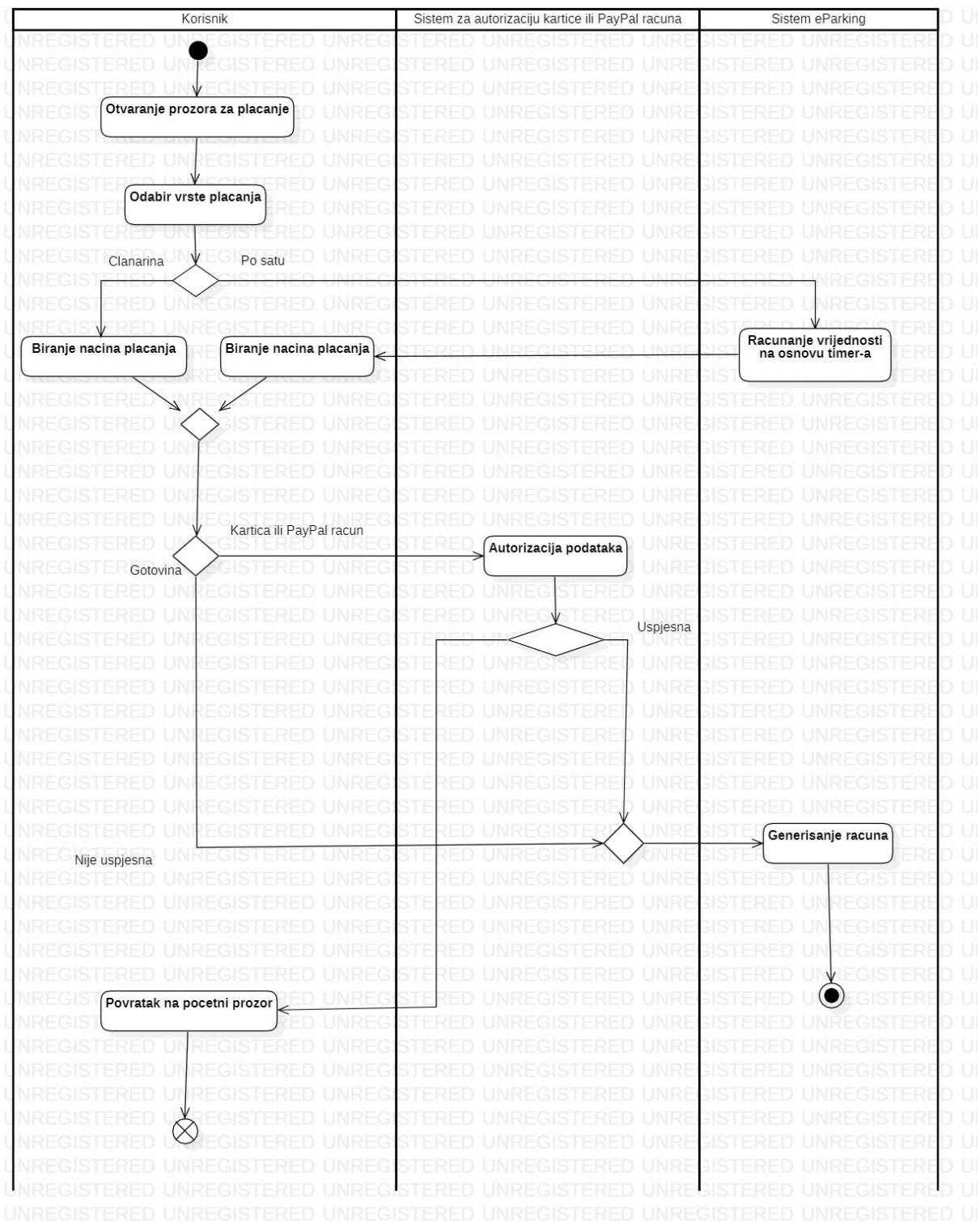
Dijagram aktivnosti za parkiranje clana:



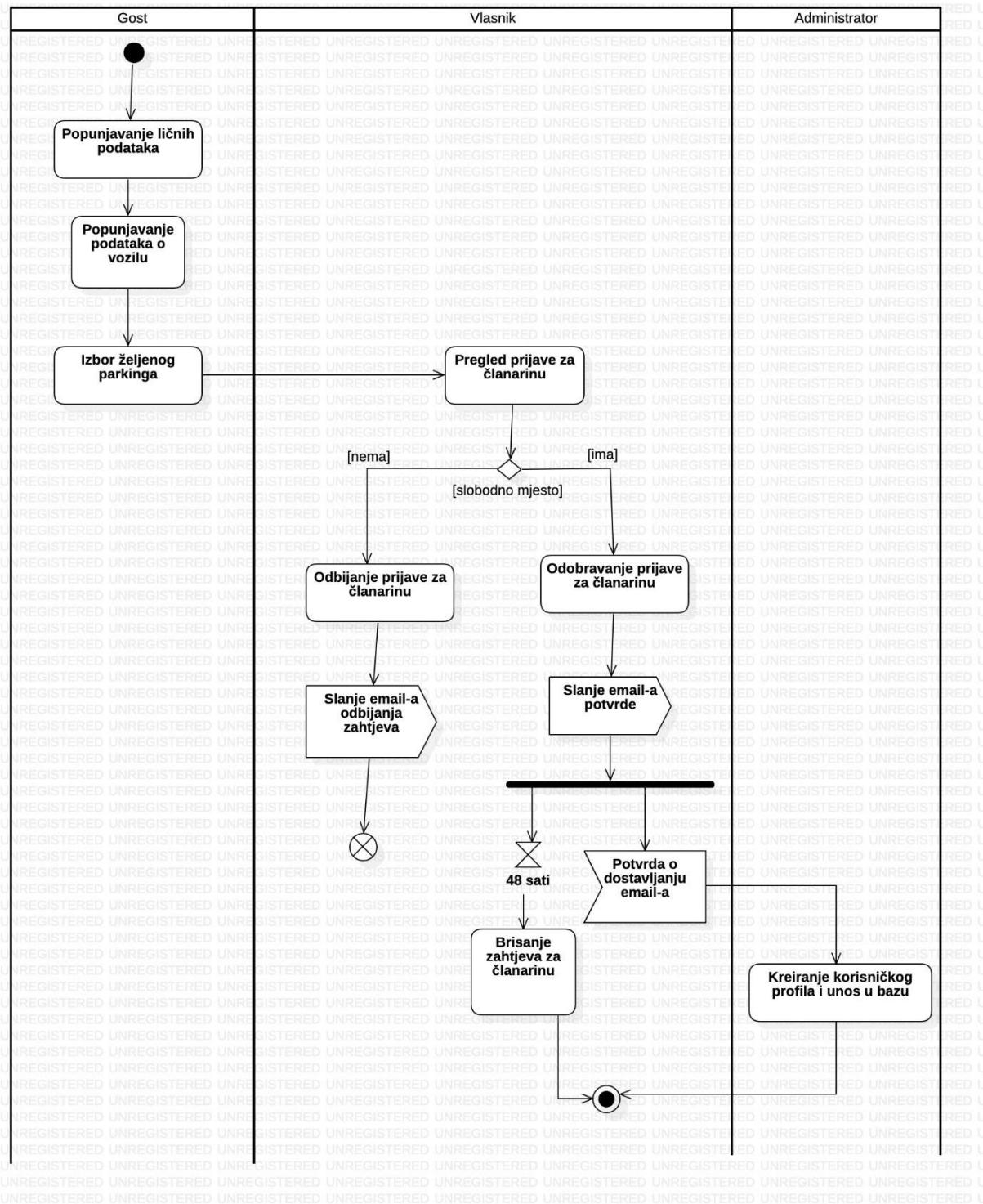
Dijagram aktivnosti za parkiranje gosta.



Dijagram aktivnosti za placanje.

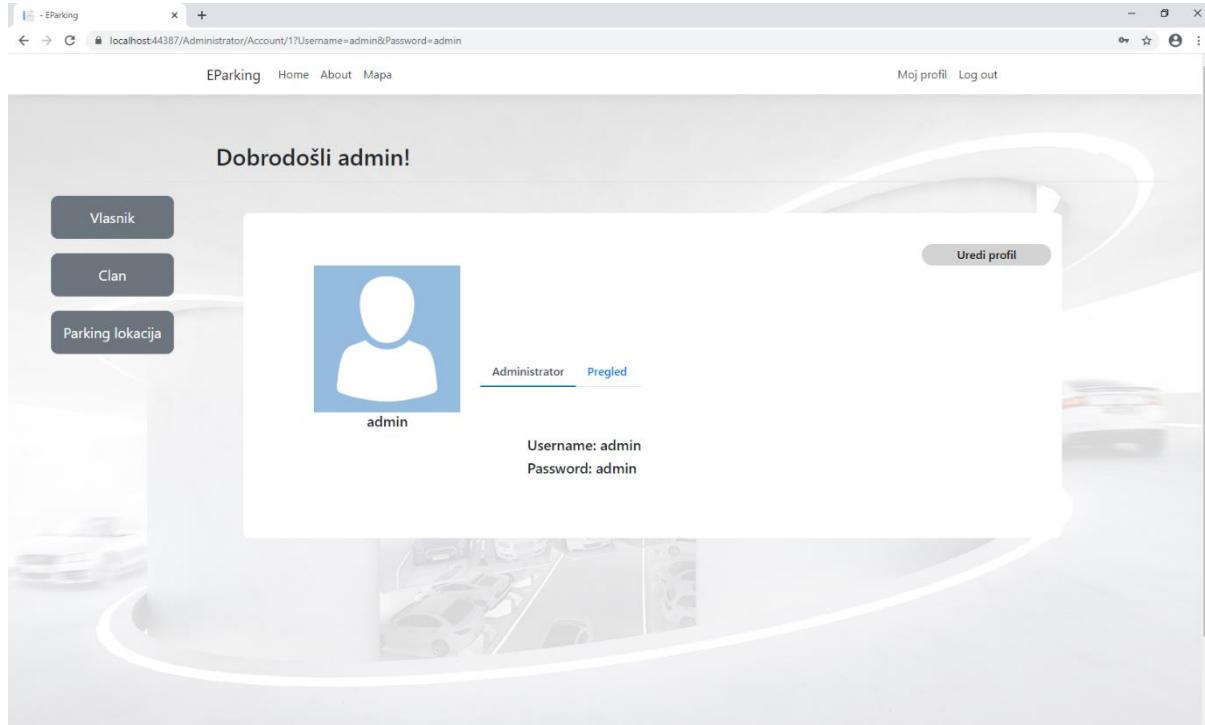


Dijagram aktivnosti za registraciju.

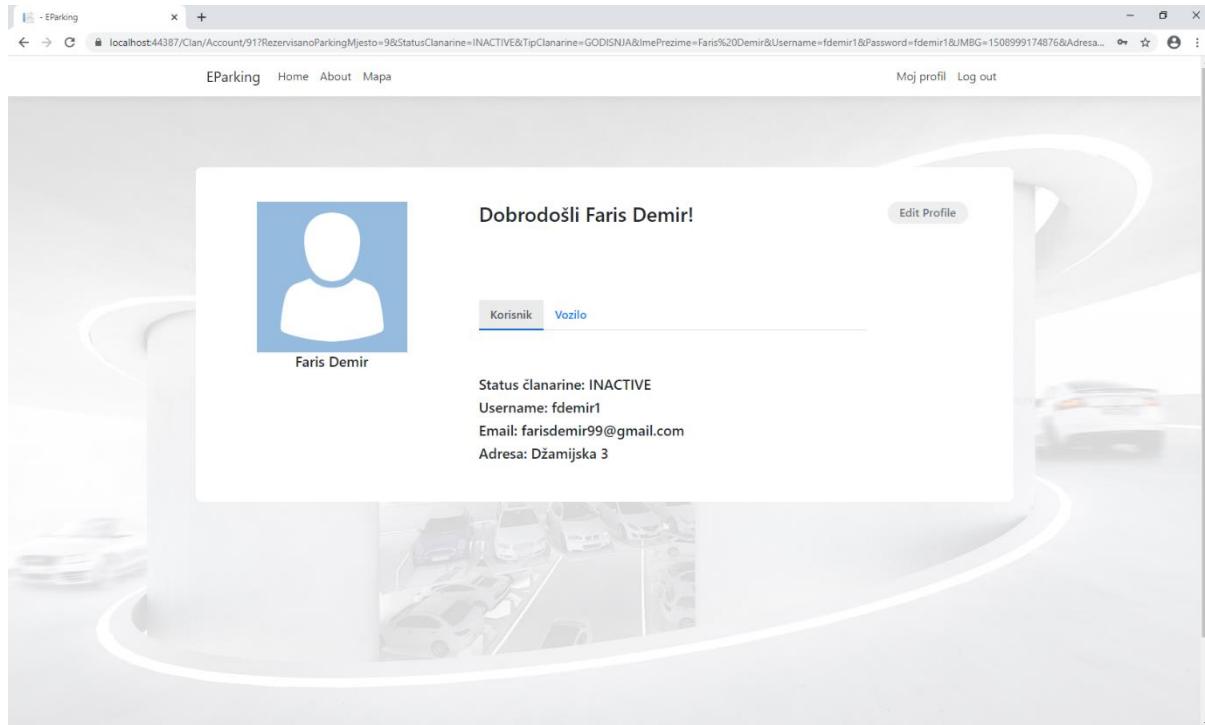


## Prototipovi grafičkih korisničkih interfejsa

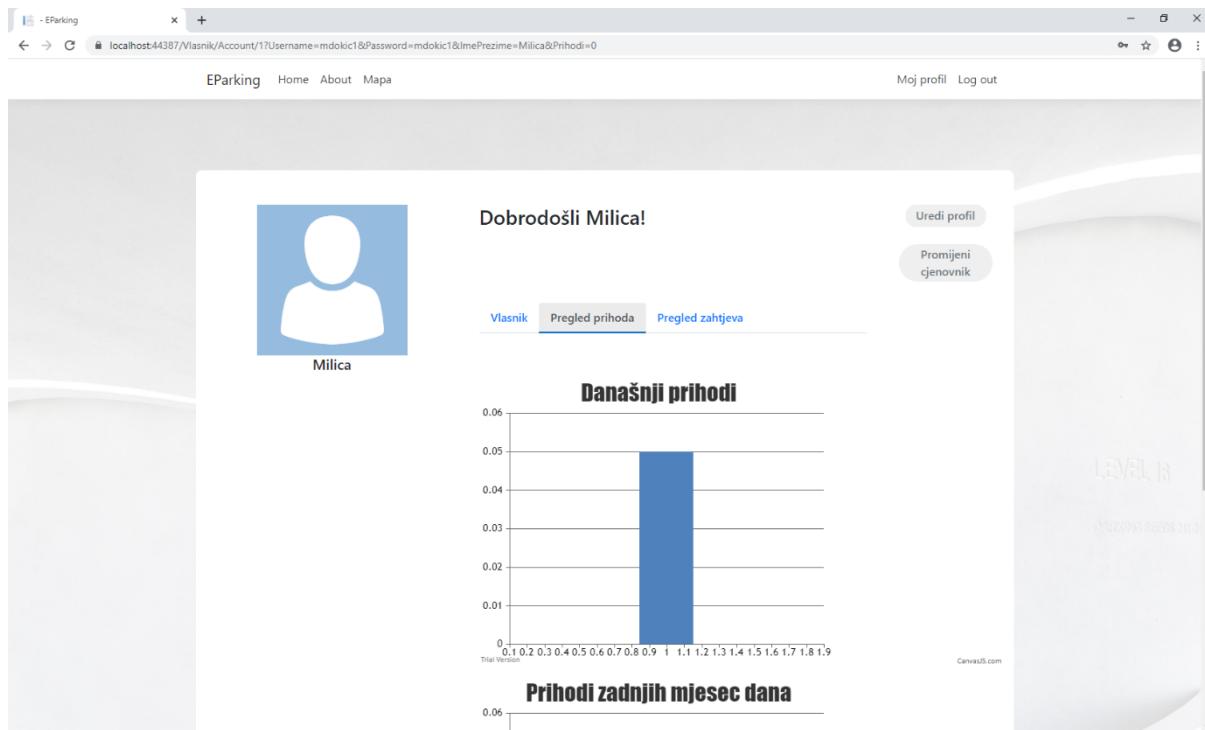
Prikaz Account-a administratora.



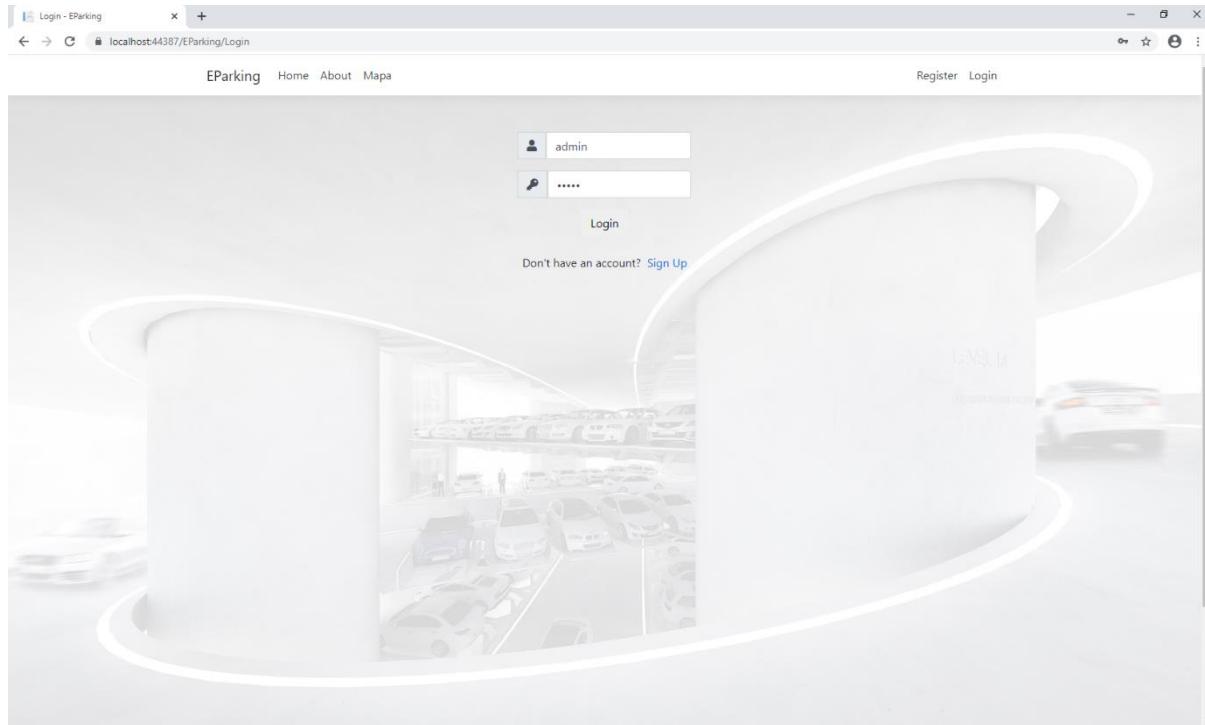
Prikaz Account-a clana.



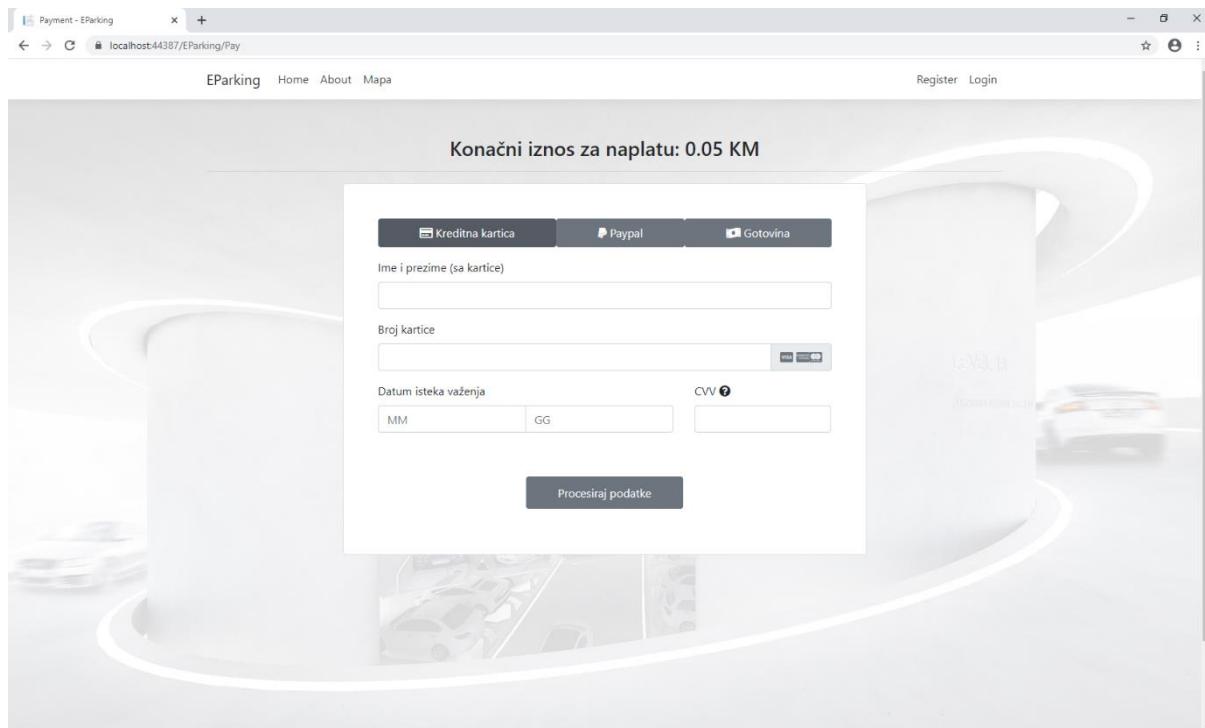
Prikaz Account-a vlasnika.



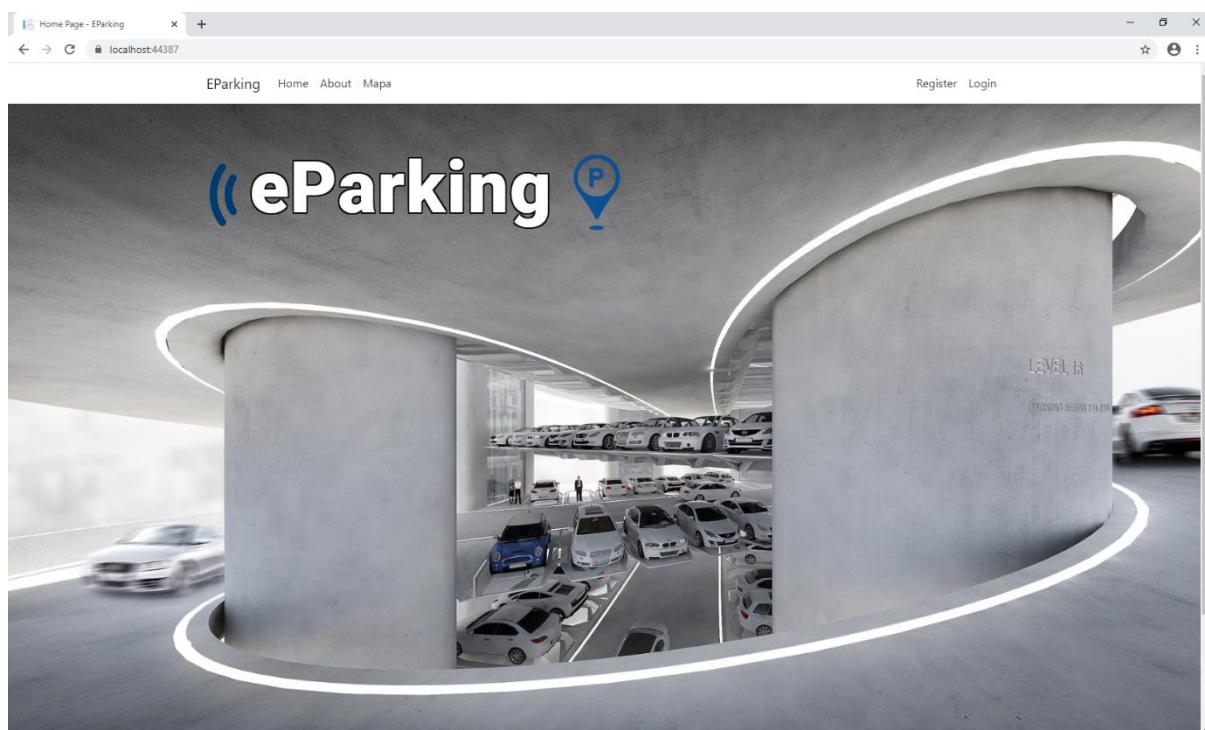
Prikaz Login prozora.



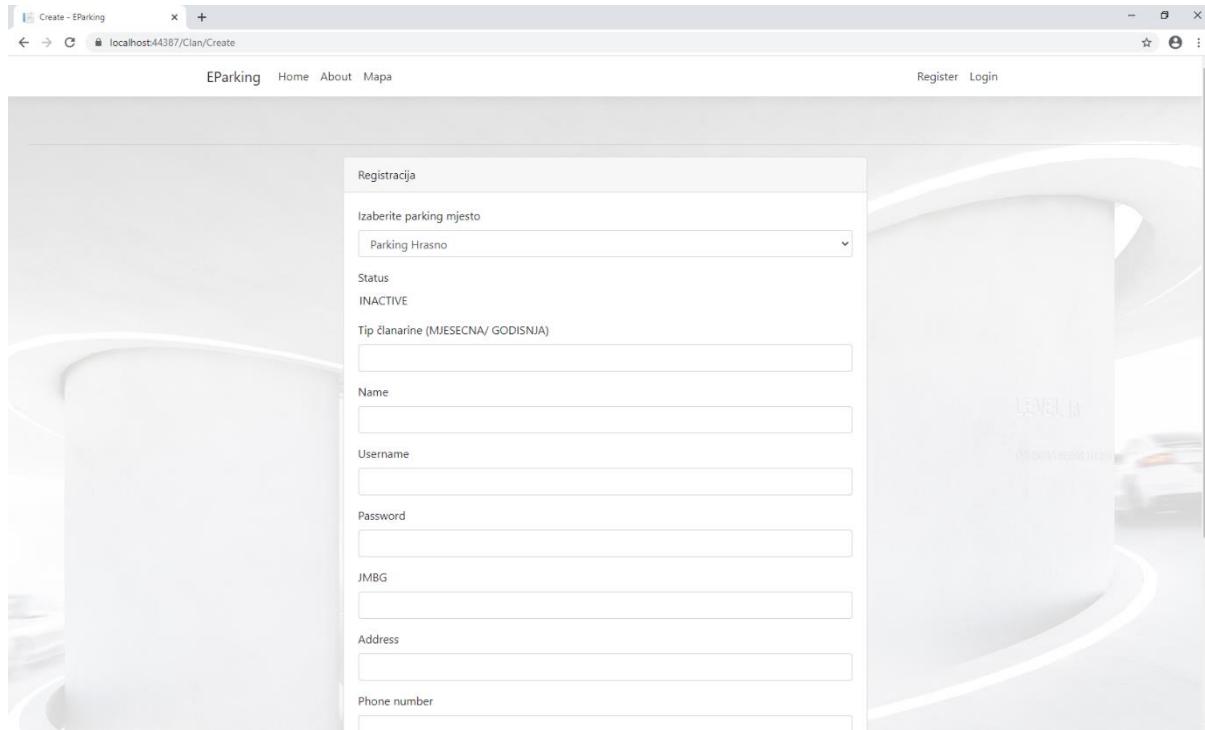
Prikaz prozora za odabir načina plaćanja.



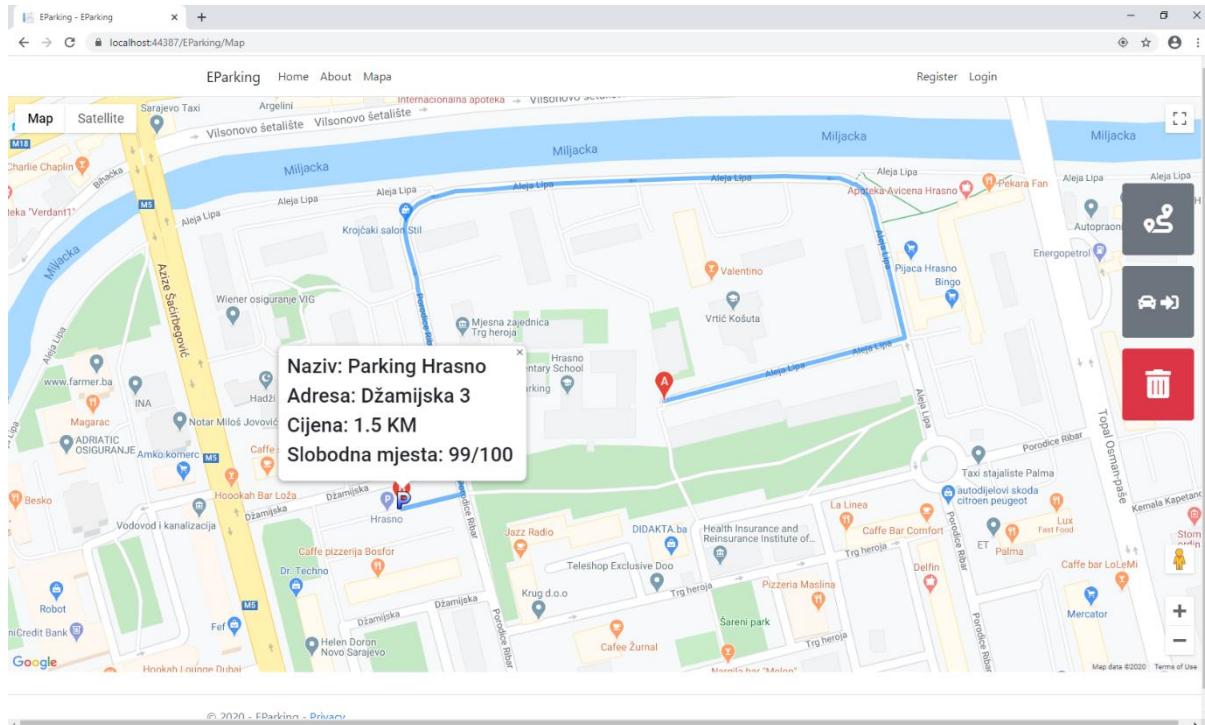
Prikaz početne strane.



## Prikaz prozora pri registraciji korisnika.



## Prikaz prozora pri traženju parkinga.



Prikaz stranice pri računanju vremena provedenog na parkingu.

The screenshot shows a web browser window titled "EParking - Timer". The URL in the address bar is "localhost:44387/EParking/Timer/0?Naziv=Parking%20Hrasno&Adresa=Džamija%203&Lat=43.849294&Long=18.3810547&Kapacitet=100&BrojSlobodnihMjesta=98&Cjenovnikid=4&Cjenovnik=EParking.Models.Cjenovnik&Vlas...". The page content is as follows:

**Dobro došli na parking!**

**Osnovne informacije**

Naziv: Parking Hrasno  
Adresa: Džamija 3  
Slobodna mjesta: 98 / 100

**Proteklo vrijeme**

**00:01:36**

**Iznos za naplatu**

**0.025 KM**

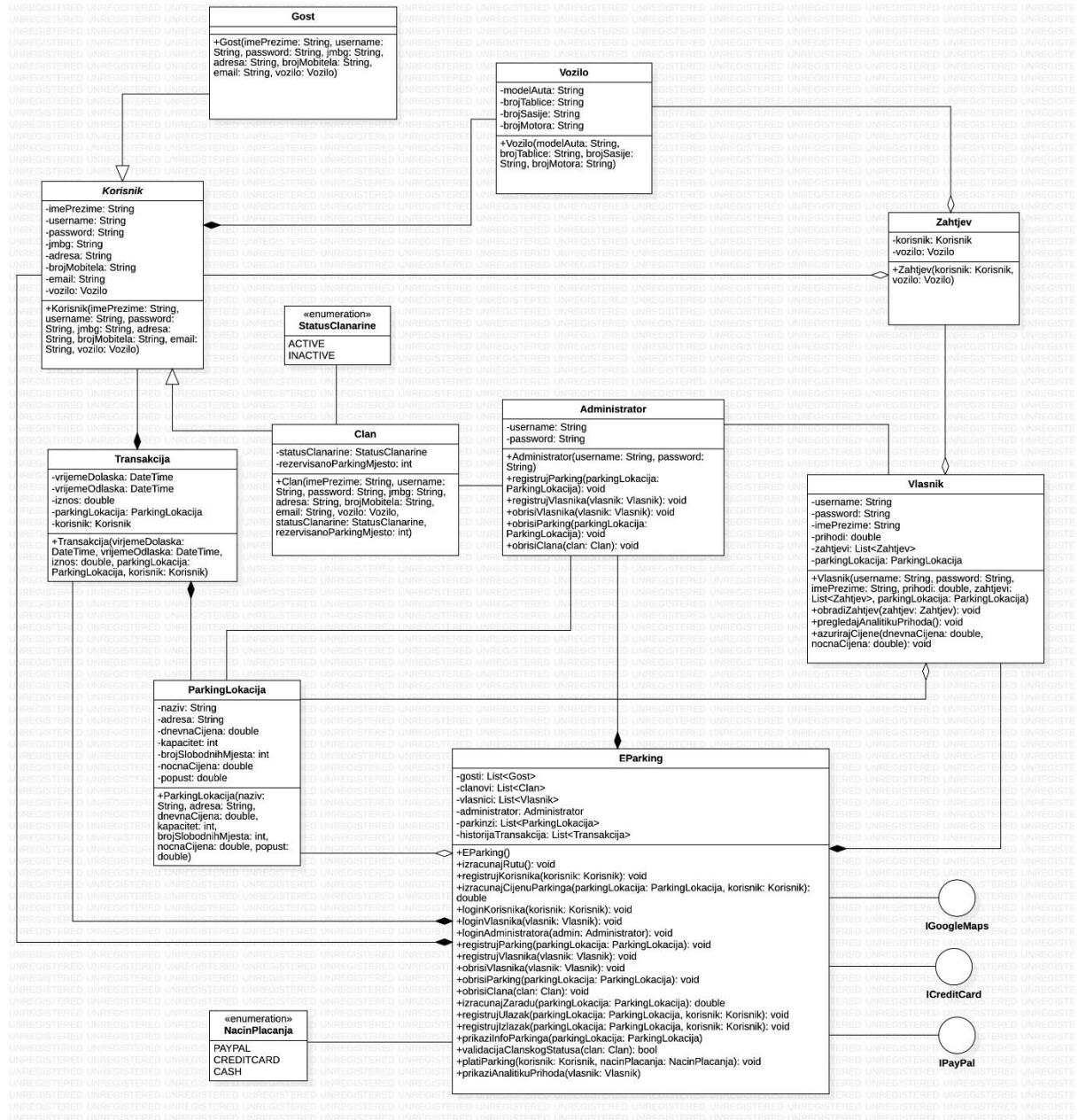
**Informacije o cijeni**

Dnevna cijena po satu: 1.5 KM  
Noćna cijena po satu: 1 KM

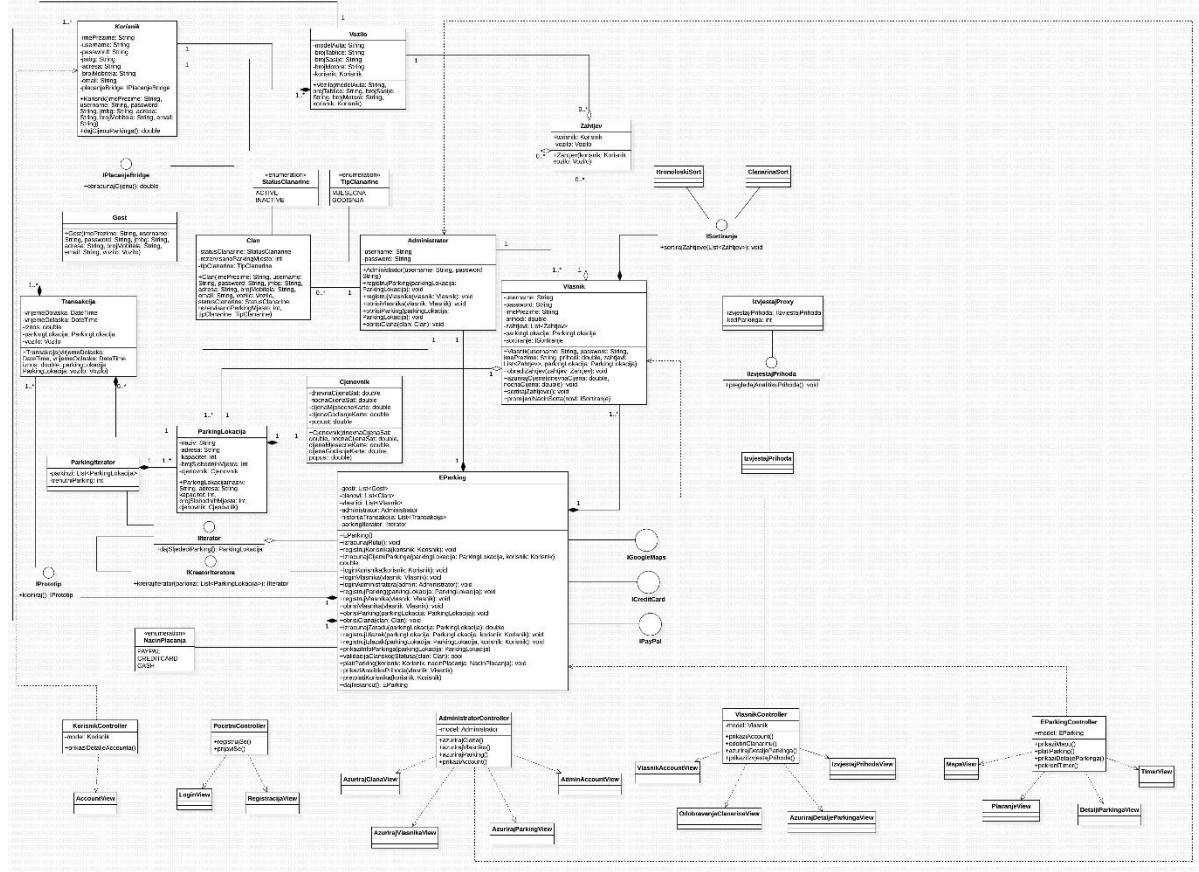
**Registruj izlazak**

The background of the page features a blurred image of a parking lot with several cars.

## Klasni dijagram (Class diagram)



## Klasni dijagram sa MVC



## SOLID principi

- **Single Responsibility Principle** – Svaka klasa treba imati samo jednu ulogu. Ovaj princip je ispoštovan, jer svaka klasa radi ono za šta je i namijenjena. Jedina koja bi mogla potencijalno narušiti ovaj princip je klasa EParking (ona vrši npr. plaćanje).
- **Open/Closed Principle** – Klasa treba biti otvorena za nadogradnje, ali zatvorena za modifikacije. Klase koje bi potencijalno mogle narušavati ovaj princip su klase EParking, Vlasnik, Korisnik, Transakcija i Zahtjev, jer klasa EParking ima atribute druge klase (Gost, Clan, Administrator, Vlasnik, ParkingLokacija, Transakcija), klasa Transakcija sadrži atribute tipa ParkingLokacija i Korisnik, klasa Vlasnik ima atribute tipa Zahtjev i ParkingLokacija, klasa Zahtjev ima atribute tipa Korisnik i Vozilo, a klasa Korisnik ima atribut tipa Vozilo. Međutim, kako ove klase ne vrše modifikaciju navedenih atributa, već ih samo koriste kao atribute (ili eventualno u listi), ovaj princip je ispoštovan.
- **Liskov Substitution Principle** – Svaka osnovna klasa treba biti zamjenjiva svim svojim podtipovima bez da to utječe na ispravnost rada programa. U našem slučaju je ovaj princip zadovoljen, jer na svim mjestima gdje koristimo baznu klasu Korisnik možemo koristiti i izvedene klase Gost i Clan.

- **Interface Segregation Principle** – Bolje je imati više specifičnih interfejsa, nego jedan generalizovani. Ovaj princip je zadovoljen, jer interface-i `IGoogleMaps`, `IPayPal` i `ICreditCard` obavljaju samo jednu vrstu akcija; `IGoogleMaps` obavlja funkciju računanja rute, `IPayPal` obavlja funkciju plaćanja pomoću PayPal računa, a `ICreditCard` obavlja funkciju plaćanja pomoću kartice.
- **Dependency Inversion Principle** – Sistem klasa i njihovo funkcionisanje treba ovisiti o apstrakcijama, a ne o konkretnim implementacijama. Ovaj princip je ispoštovan, jer je bazna klasa `Korisnik` ujedno i apstraktna klasa iz koje su izvedene klase `Gost` i `Clan`.

## Paterni

### STRUKTURALNI PATERNI

#### 1. Adapter patern

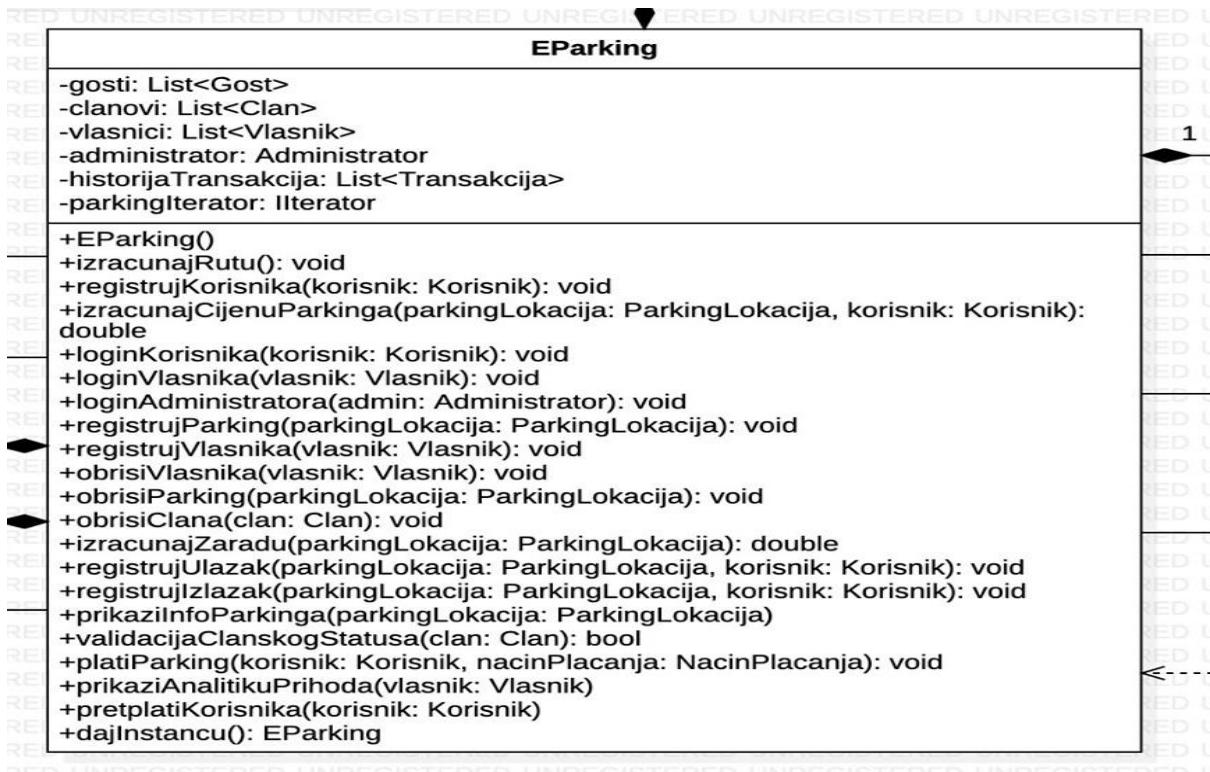
Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima.

Ukoliko bismo uveli korištenje loyalty kartice na kojoj se skupljaju bodovi, onda bi mogli iskoristiti Adapter patern, da pretvorimo te bodove u popust.

#### 2. Facade patern

Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.

*EParking* nam je fasada, unutar koje se obavljaju procesi u koje korisnik nema uvid (npr. plaćanje preko metode `platiParking` koja u sebi poziva nake druge implementirane metode interfejsa `IPayPal` i `ICreditCard`; parkiranje preko metode `izracunajRutu` koja poziva neke implementirane metode interfejsa `IGoogleMaps`).



### 3. Decorator patern

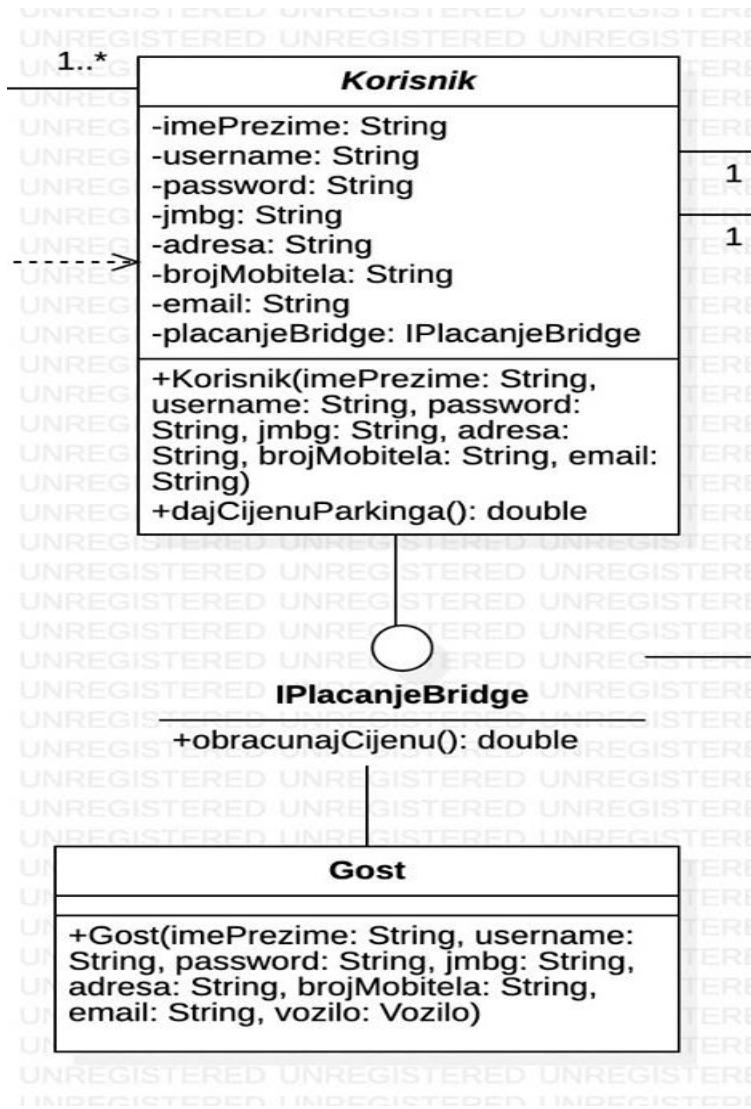
Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata.

Trenutno nemamo potrebu za primjenom ovog paterna. Ukoliko bi se odlučili da primijenimo ovaj patern, mogli bismo dodati mogućnost slanja obavještenja ne samo preko E-Maila, nego i preko SMS-a, Facebook-a, Viber-a, WhatsApp-a.

### 4. Bridge patern

Bridge patern služi kako bi se apstrakcija nekog objekta od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

U našem sistemu, primjenili smo Bridge patern kod obračuna cijena u zavisnosti od tipa korisnika. Gost plaća cijenu po satu, a član u zavisnosti od tipa članarine. Kreirali smo interface *IPlaćanjeBridge* koji u sebi sadrži metodu *obracunajCijenu*, i tu metodu svaka od klasa implementira po svojim potrebama.



## 5. Composite patern

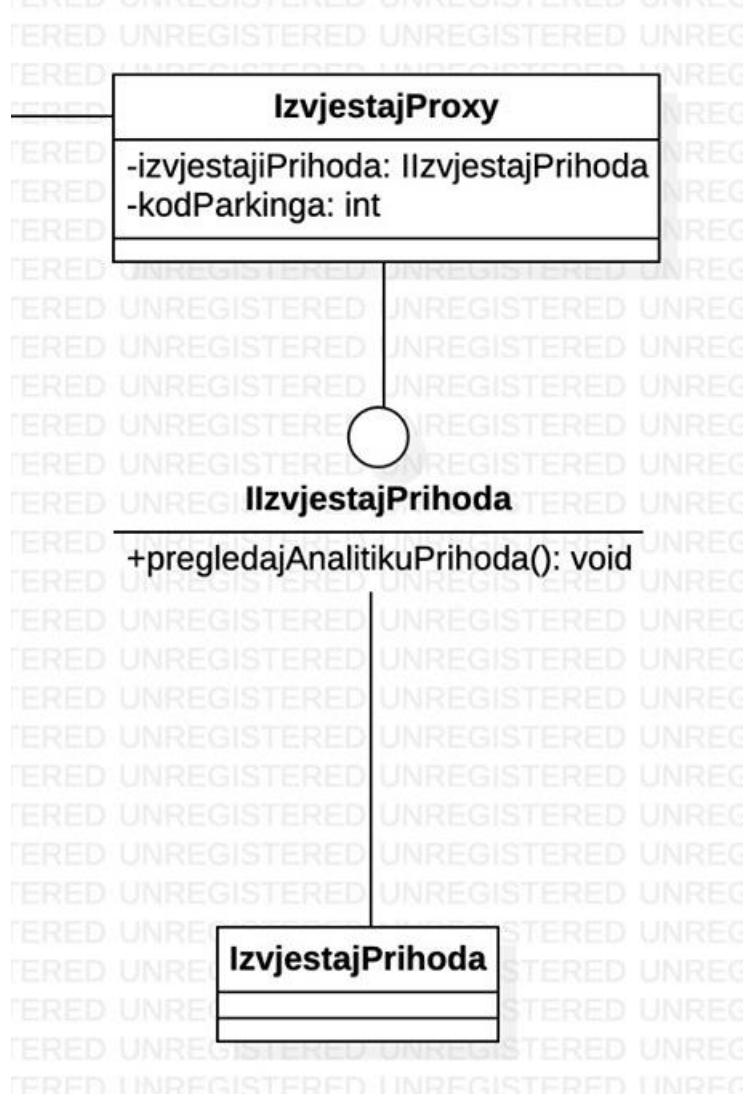
Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.

Ako bi u našem sistemu, klasu *Administrator* učinili apstraktnom i iz nje naslijedili klase *AdministratorParkinga* i *AdministratorKorisnika*, tada bi dodali interface *IAdministrator* sa metodom *azuriraj*, koju bi svaka od izvedenih klasa implementirala na svoj način.

## 6. Proxy patern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

U našem sistemu primijenili smo Proxy patern kod pregleda analitike prihoda od strane vlasnika parkinga. Dodali smo klasu *IzvjestajProxy* koja u sebi sadrži instancu interfejsa *IizvjestajProxy* koja ima metodu *pregledajAnalitikuPrihoda*. Na osnovu atributa *kodParkinga* ograničeno je da svaki vlasnik može pregledati analitiku prihoda svog parkinga.



## 7. Flyweight patern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta (tzv. bezlično stanje). Korištenje ovog patterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije.

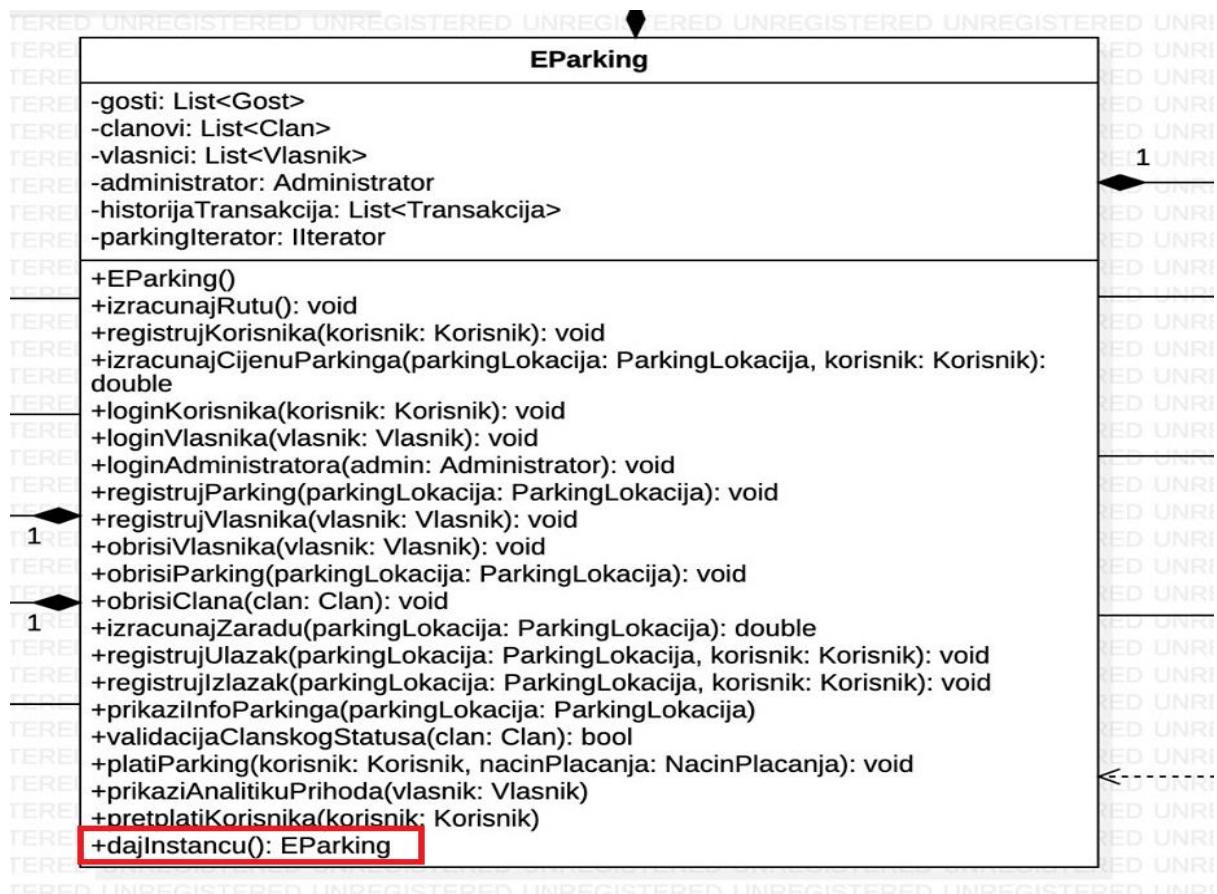
Nismo implementirali ovaj patern, ali mogli bismo ga primijeniti kada bismo dodali klasu *ClanFirme*, u slučaju da se svi radnici neke firme parkiraju na istom parkingu i samim tim imaju određeni popust. Kada bi se član neke firme prijavljivao na sistem, ne bismo svaki put kreirali novu instancu klase *ClanFirme*, već bi se instancirao samo jedan objekat ovog tipa i tako bismo uštedili memoriju.

## KREACIJSKI PATERNI

### 1. Singleton patern

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa.

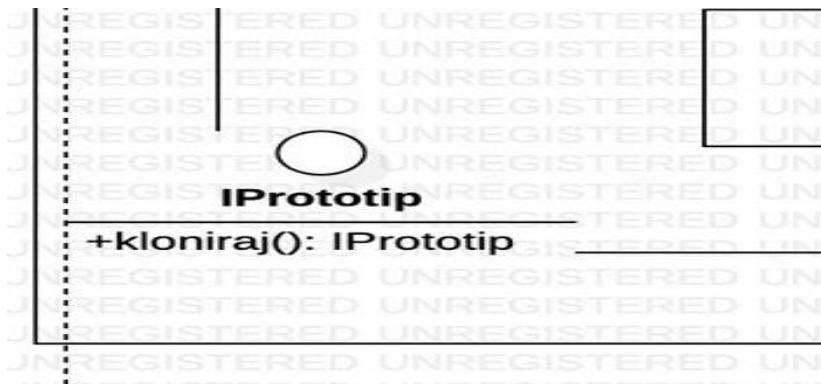
Ovaj patern je primijenjen na klasi EParking, jer je potrebna samo jedna instanca ove klase. Dodali smo metodu dajInstancu u klasu EParking.



### 2. Prototype patern

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti.

Mi smo u našem sistemu implementirali ovaj patern, tako što smo dodali interface *IPrototip*, koji ima metodu *kloniraj*. Time smo pojednostavili proces kreiranja novih instanci objekta, a klasa koja ga implementira je *Transakcija*.



### 3. Factory Method patern

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijedenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način.

Mi nismo u našem sistemu implementirali ovaj patern. Ukoliko bismo ga željeli implementirati, mogli bismo dodati klasu *AnalitikaPrihoda* iz koje su naslijedene klase *GodisnjiPrihodi* i *MjesecniPrihodi*, a klasa *IzvjestajPrihoda* bi u sebi imala atribut *analitikaPrihoda*. Dodali bismo i dvije izvedene klase *StarIzvjestajPrihoda* i *NovIzvjestajPrihoda* (izvedene iz klase *IzvjestajPrihoda*), koje implementiraju interface *IIzvjestaj* koji sadrži metodu *kreirajIzvjestaj* (factory metoda).

### 4. Abstract Factory patern

Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasleđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati.

U našem sistemu nismo implementirali ovaj patern. Navedeni patern bismo mogli implementirati ukoliko bismo dodali interface *IFactory* sa metodom *dajNacinPlacanja*, čime bismo smanjili broj if-else uslova za određivanje načina plaćanja. Mi smo trenutno to rješili korištenjem enumeracije *NacinPlacanja*.

### 5. Builder patern

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa.

Trenutno nismo implemenitrali ovaj patern. Ukoliko bismo uzimali u obzir i parkinge za veća vozila (npr. kamione , autobuse) , tada bismo mogli omogućiti da se na početku bira tip vozila, i

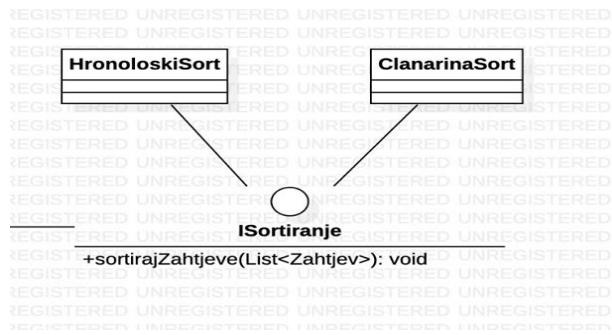
na mapi bi se prikazale informacije samo o tim parkinzima. Dodali bismo i interface *IBuilder* koji sadrži metode koje će izvršavati prikaz samo određenih parkinga, kao i klasu *TipVozilaBuilder* koja implementira metode interfejsa.

## PATERNI PONAŠANJA

### 1. Strategy patern

Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji.

U našem sistemu smo implementirali ovaj patern tako što smo uveli sortiranje zahtjeva korisnika koje odobrava vlasnik. Dodali smo interface *ISortiranje*, i klase *HronoloskiSort* i *ClanarinaSort* koje predstavljaju sortiranje na osnovu zahtjeva korisnika po redu kako su pristizali, i sortiranje na osnovu tipa članarine.



### 2. State patern

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi.

Ovaj patern nismo implementirali u našem sistemu, ali ukoliko bismo to htjeli uraditi, mogli bismo dodati interface *IStatusClanarine*, i klase *AktivnaClanarina* i *NeaktivnaClanarina* koje bi implementirale taj interface, i metodu *platiClanarinu* kojom bi status članarine iz neaktivnog prelazio u aktivni.

### 3. Template Method patern

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine.

U našem sistemu nismo implementirali ovaj patern, ali mogli bismo to uraditi tako što uredimo klasu *Sortiranje*, kao i klase *AbecedniSort*, *SlobodnaMestaSor* koje predstavljaju sortiranje parking lokacija prilikom prikaza po abecedi, po broju slobodnih mesta i po posjećenosti parkinga.

## 4. Observer patern

Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Preplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama.

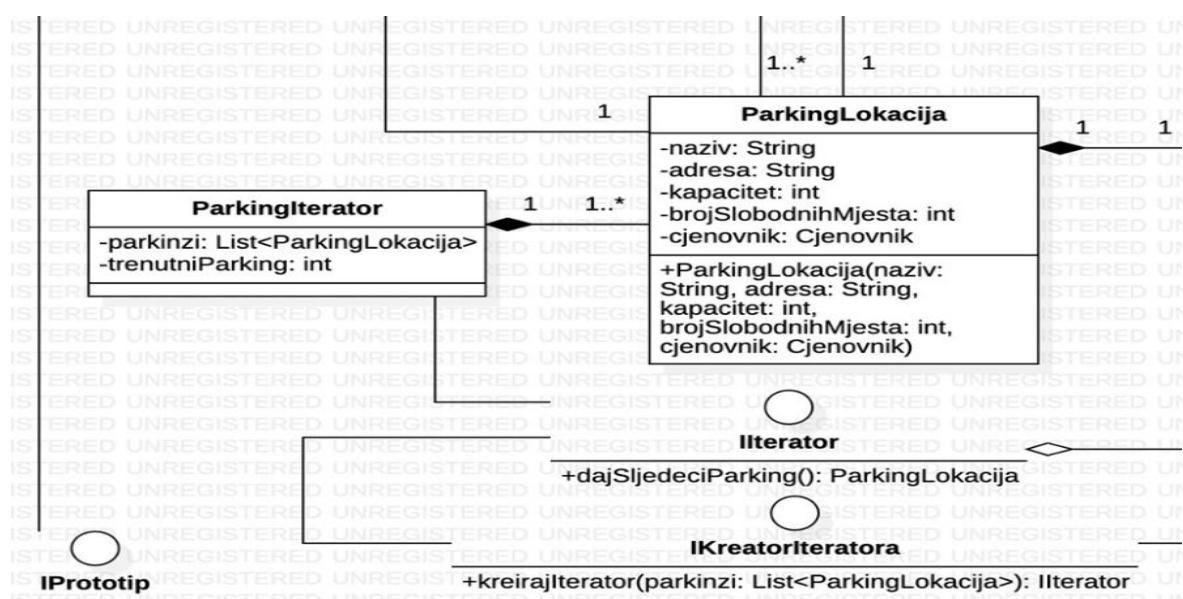
Ovaj patern nismo implementirali, ali mogli bismo to uraditi tako što bismo dodali mogućnost da korisnici koji imaju plaćenu članarinu primaju notifikacije o nekim promjenama vezanim za parking za koji su registrovani (npr. promjena kapaciteta, cijena i sl.). To bismo postigli tako što bismo dodali interface *IObserver* koji će imati metodu *posaljiObavjestenje*. U klasi EParking nam treba lista svih korisnika koji su platili članarinu, kao i metoda *obavijestiClanove*.

## 5. Iterator patern

Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste.

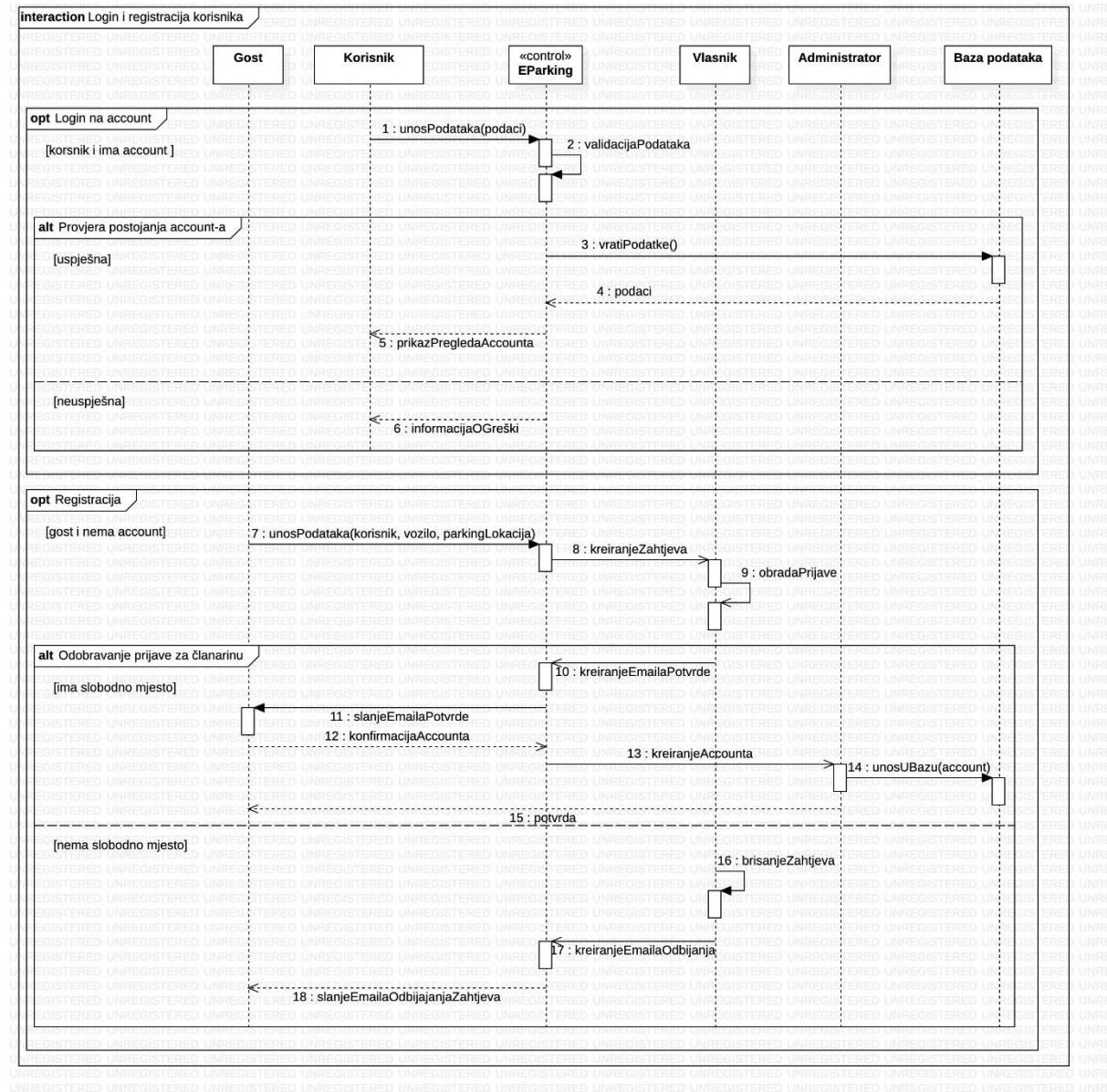
Izvedba liste može biti u obliku drveta, jednostrukе liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija.

Ovaj patern smo implementirali u našem sistemu, tako što smo omogućili prolazak kroz listu parking lokacija. Dodali smo interfejs *IKreatorIteratora* sa metodom *kreirajIterator*, interfejs *Iterator* sa metodom *dajSljedeciParking*, i klasu *ParkingIterator*, u kojoj se nalazi lista parking lokacija. U klasi *EParking* nam nije više potrebna lista parking lokacija, već dodajemo *ParkingIterator*.

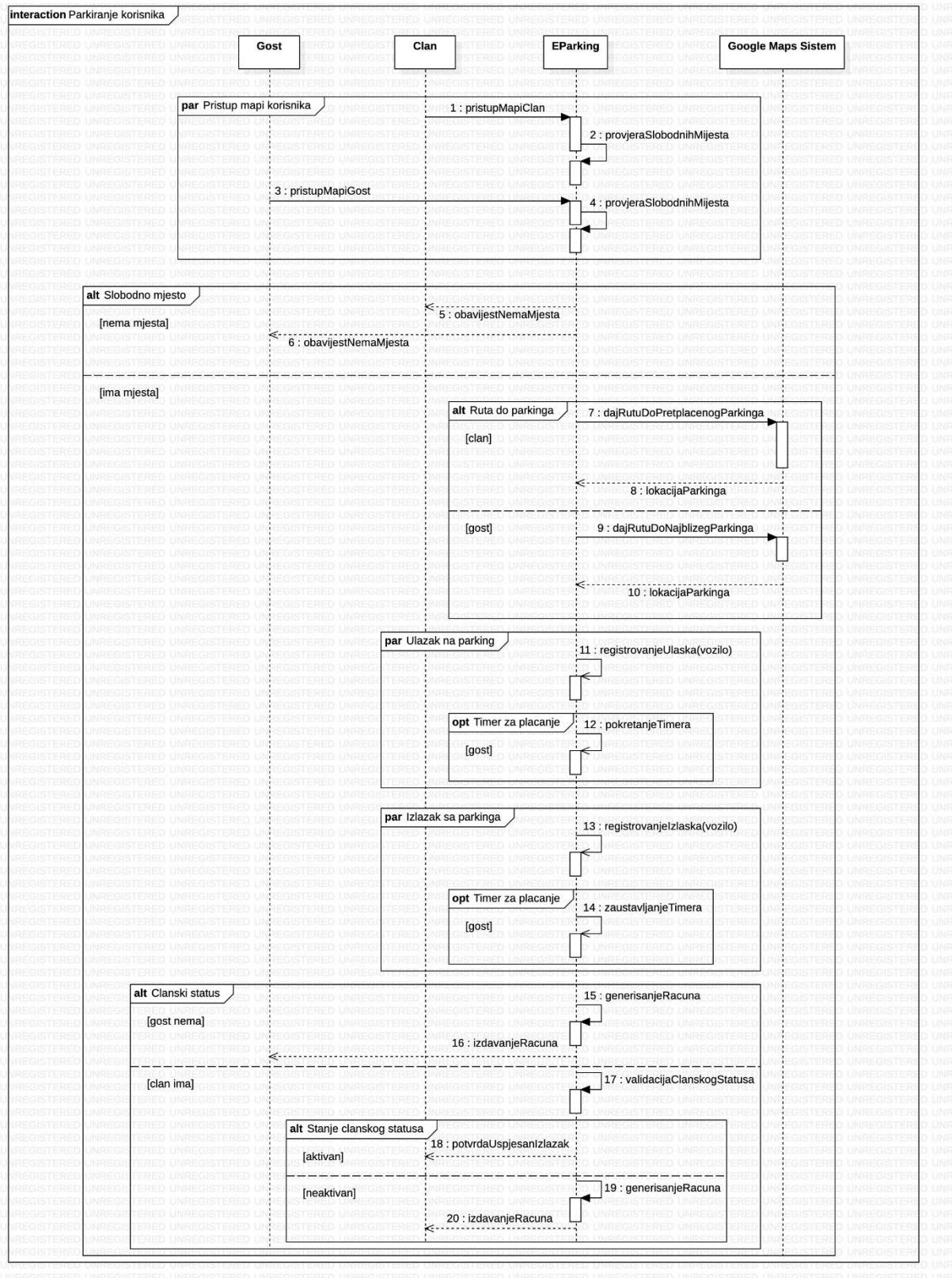


## Dijagrami skvence (Sequence diagrams)

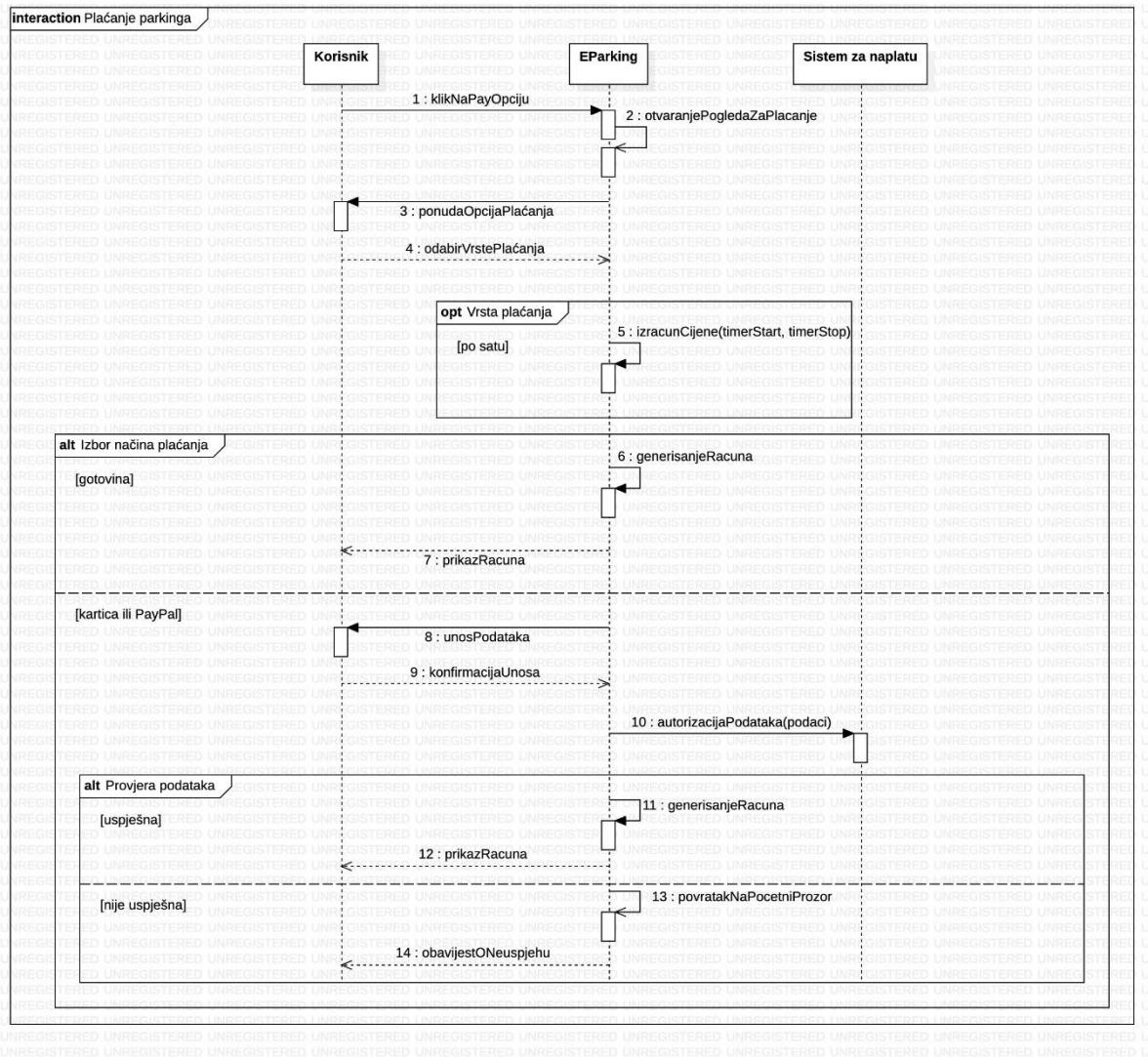
Dijagram sekvence za login i registraciju korisnika.



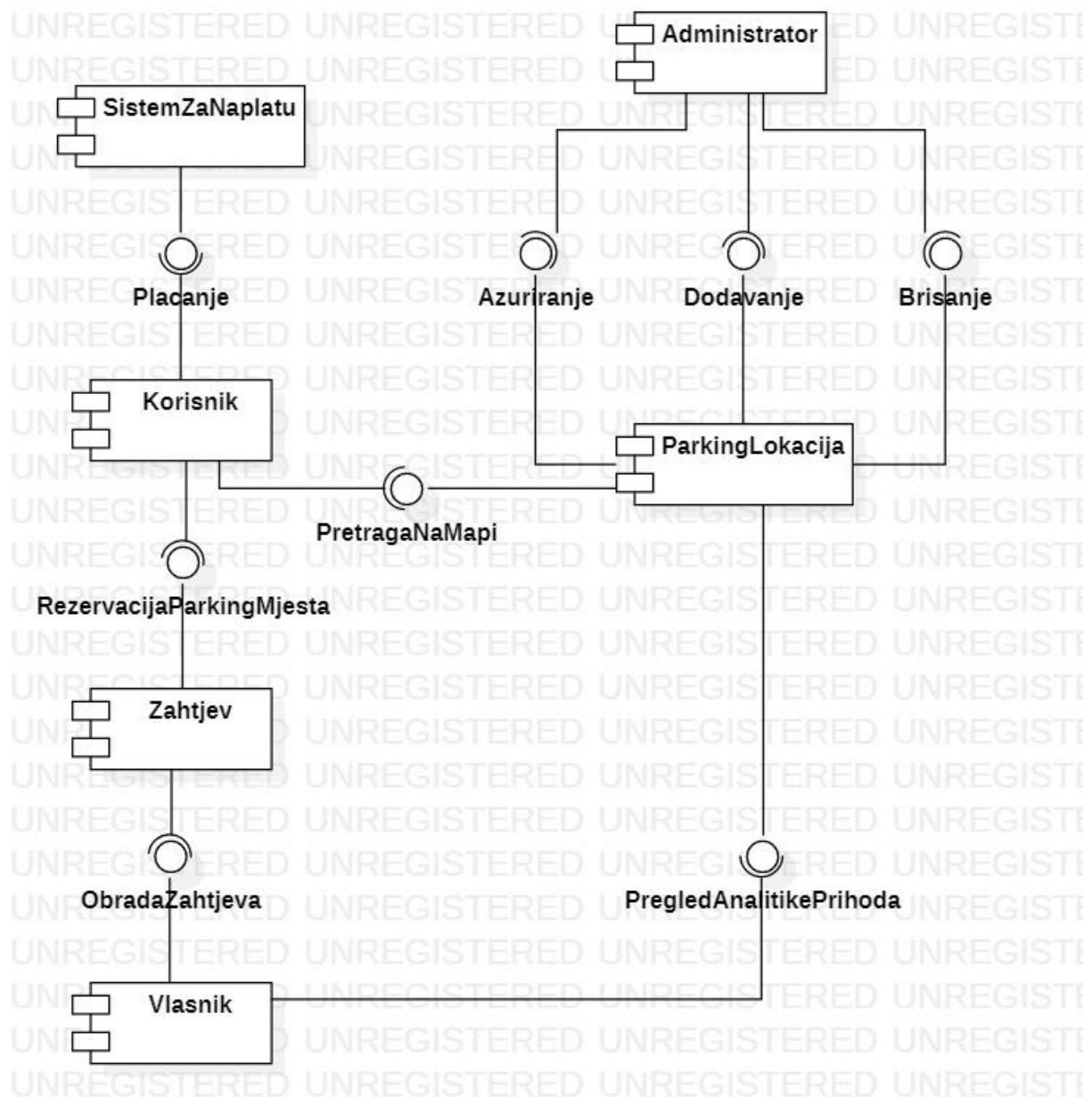
## Dijagram za parkiranje korisnika.



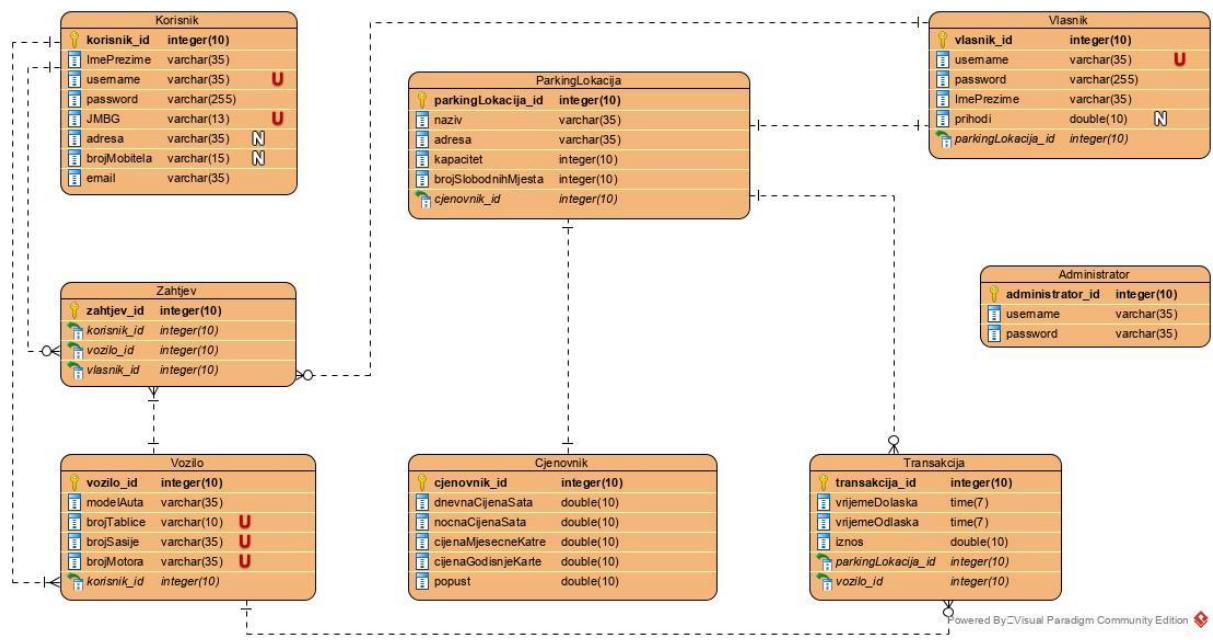
## Dijagram za placanje parkingu.



## Dijagram komponenti (Component diagram)



## ER dijagram



## Web servisi

API (eng. Application Programming Interface) predstavlja mehanizam za pružanje usluga drugim programima. Web servise je jedna vrsta API i najčešće se realizuje putem HTTP protokola. Web servis koji ćemo koristiti u našoj aplikaciji je google maps (<https://www.google.com/maps>). Koristeći google maps, korisnicima naše aplikacije dajemo mogućnost da nadju najkraću rutu od svoje trenutne lokacije do najbližeg parkinga ili da se preračuna ruta od trenutne lokacije do najbližeg parkinga koji ima slobodnih mesta.